OpenEuler进程创建与变量独立性实验

获取PID实验

1.创建源代码文件

```
vi yi.cpp
```

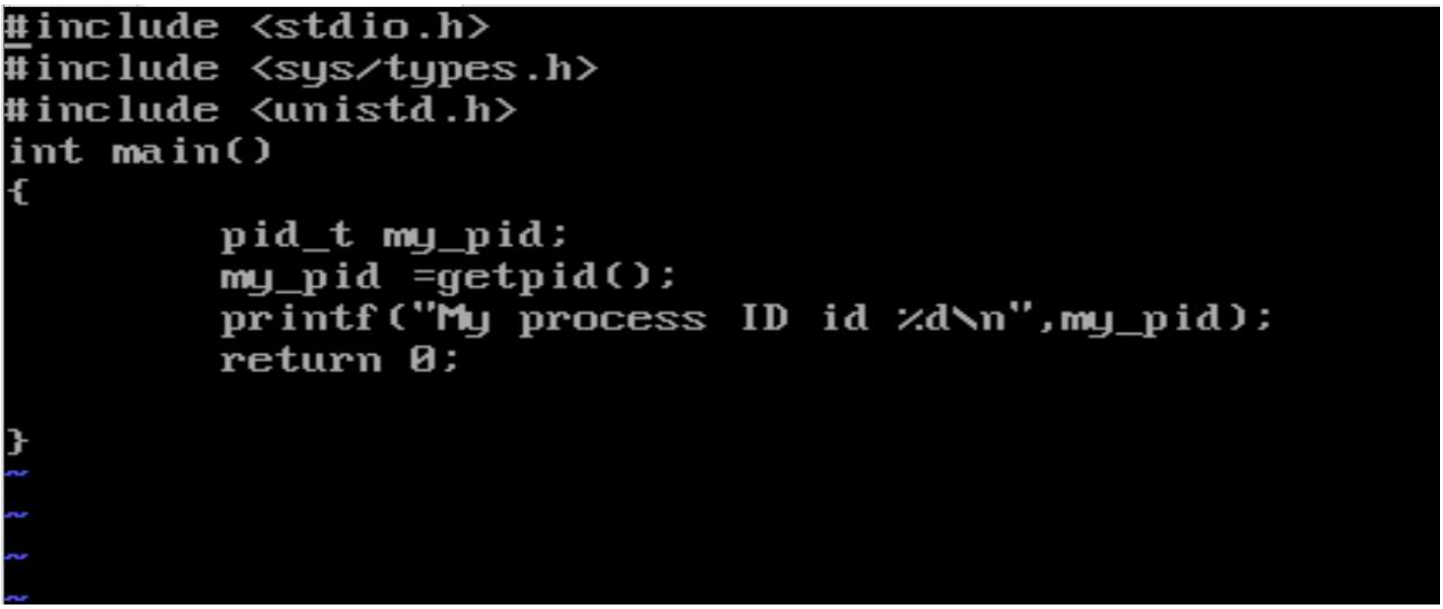
2.进入文件编写代码

进入文件以后按“a”键进入编辑模式
在yi.cpp中编写以下代码
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

```
int main()
{
    pid_t my_pid;
    my_pid = getpid();
    printf("My process ID is %d\n", my_pid);

    return 0;
}
```

如图所示:



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        pid_t my_pid;
        my_pid =getpid();
        printf("My process ID id %d\n",my_pid);
        return 0;

}
~
~
~
~
```

按 ESC 退出编辑模式
按住 `shift + :` 并输入wq
按下回车键退出文件

编译并运行代码

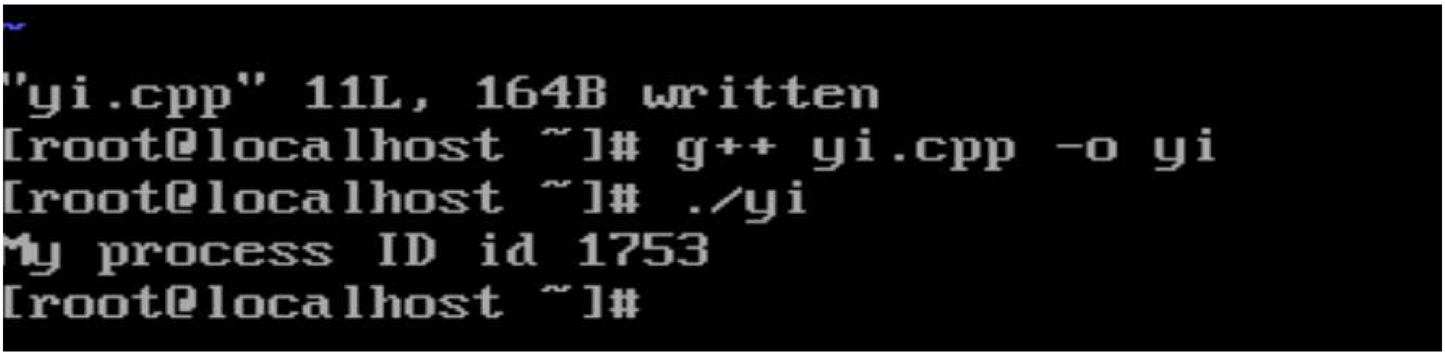
使用如下代码编译代码

```
g++ yi.cpp -o yi
```

运行程序

```
./yi
```

输出结果如图所示：



```
~
"yi.cpp" 11L, 164B written
[root@localhost ~]# g++ yi.cpp -o yi
[root@localhost ~]# ./yi
My process ID id 1753
[root@localhost ~]#
```

获取到的当前进程号为1753

进程创建与父子进程关系实验

1.创建源代码文件

创建文件 er

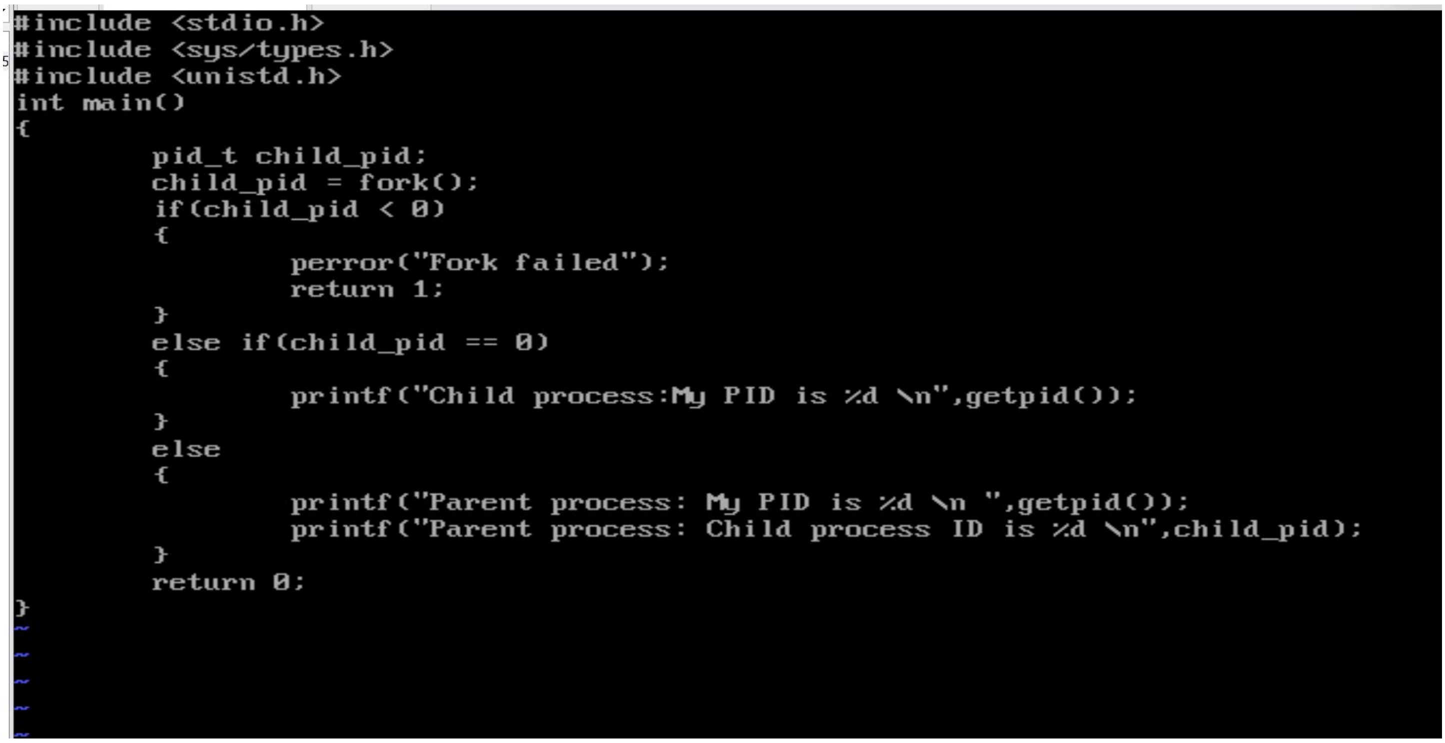
```
vi er.cpp
```

2.输入代码

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int main（）
{
    pid_t child_pid;
    child_pid fork();
    if( child_pid < 0 )
    {
        perror("Fork failed");
        return 1;
    }
    else if( child_pid == 0 )
    printf("Child process:My PID is %d \n",getpid() );
    else
    {
        printf ("Parent process:Child Process ID is %d \n ",child_pid);
        int status;
        waitpid(child_pid,&status,0);
        if (WIFEXITED(status))
        printf ('Parent process:Child exited with status %d \n",WEXITSTATUS(statu
    }

    return 0:
}
```

如图所示:



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        pid_t child_pid;
        child_pid = fork();
        if(child_pid < 0)
        {
                perror("Fork failed");
                return 1;
        }
        else if(child_pid == 0)
        {
                printf("Child process:My PID is %d \n",getpid());
        }
        else
        {
                printf("Parent process: My PID is %d \n ",getpid());
                printf("Parent process: Child process ID is %d \n",child_pid);
        }
        return 0;
}
~
~
~
~
~
```

3.编译并运行代码

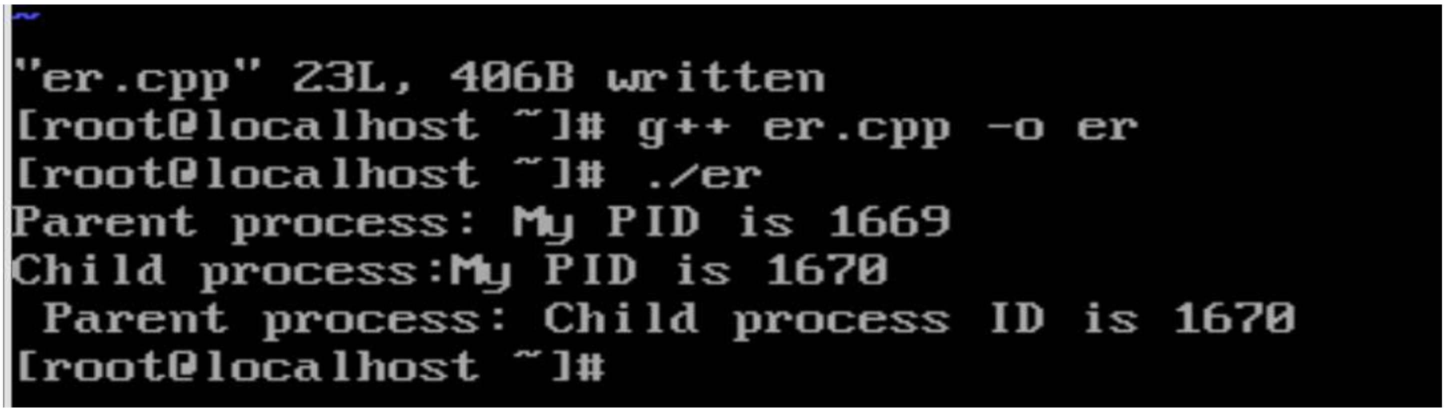
编译代码

```
g++ er.cpp -o er
```

运行程序

```
./er
```

输出结果如图所示：



```
~
"er.cpp" 23L, 406B written
[root@localhost ~]# g++ er.cpp -o er
[root@localhost ~]# ./er
Parent process: My PID is 1669
Child process:My PID is 1670
 Parent process: Child process ID is 1670
[root@localhost ~]#
```

`fork()` 执行成功以后父进程会产生一个子进程
父进程会输出自己的进程号和子进程号，而子进程只输出自己进程号

父进程等待子进程退出测试

1.修改 `er.cpp` 的代码

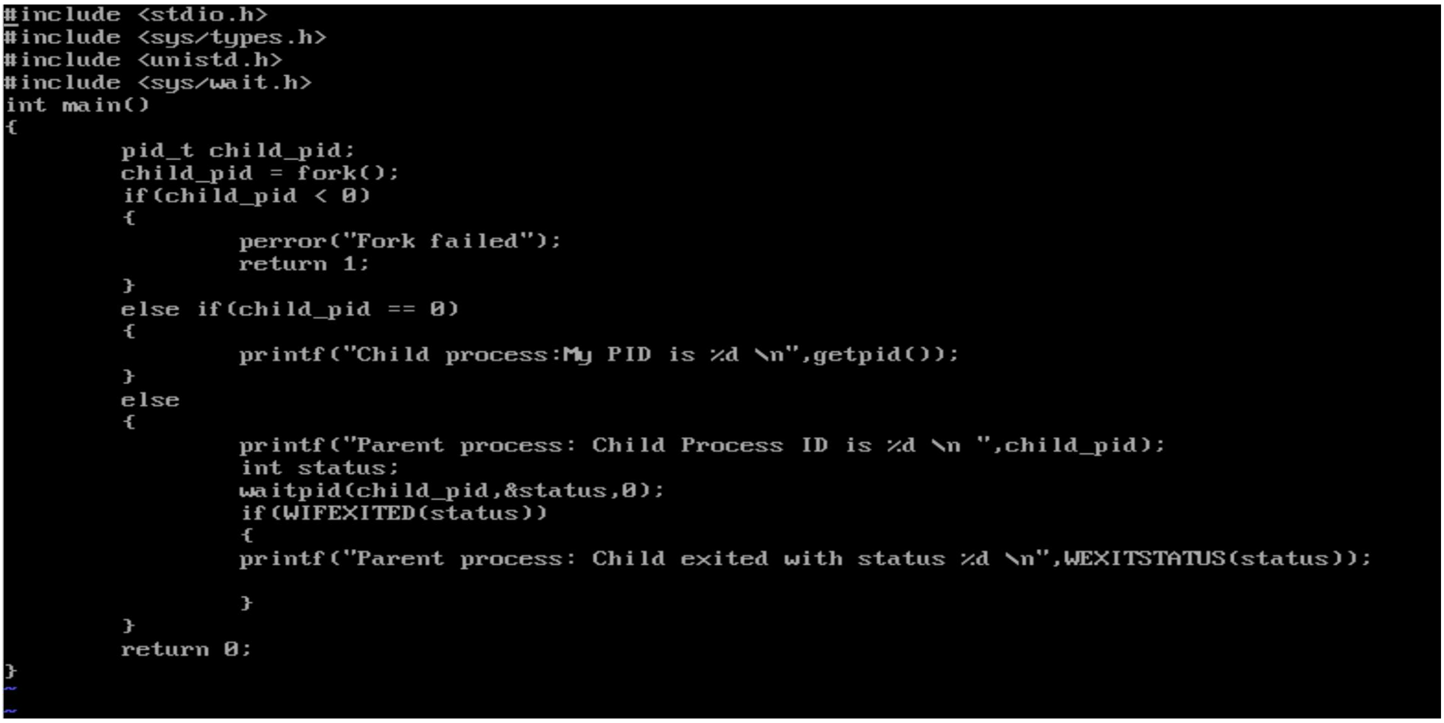
```
vi er.cpp
```

修改为以下代码

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    pid_t child_pid;
    child_pid = fork();
    if (child_pid < 0)
    {
        perror("Fork failed");
        return 1;
    }
    else if (child_pid == 0)
    {
        printf("Child process:My PID is %d \n", getpid());
    }
    else
    {
        printf("Parent process: Child process ID is %d \n", child_pid);
        int status;
        waitpid(child_pid, &status, 0);
        if (WIFEXITED(status))
        {
            printf("Parent process: Child exited with status %d\n", WEXITSTATUS(
        }
    }
    return 0;
}
```

如图所示:



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
        pid_t child_pid;
        child_pid = fork();
        if(child_pid < 0)
        {
                perror("Fork failed");
                return 1;
        }
        else if(child_pid == 0)
        {
                printf("Child process:My PID is %d \n",getpid());
        }
        else
        {
                printf("Parent process: Child Process ID is %d \n ",child_pid);
                int status;
                waitpid(child_pid,&status,0);
                if(WIFEXITED(status))
                {
                printf("Parent process: Child exited with status %d \n",WEXITSTATUS(status));

                }
        }
        return 0;
}
~
~
```

2.运行代码

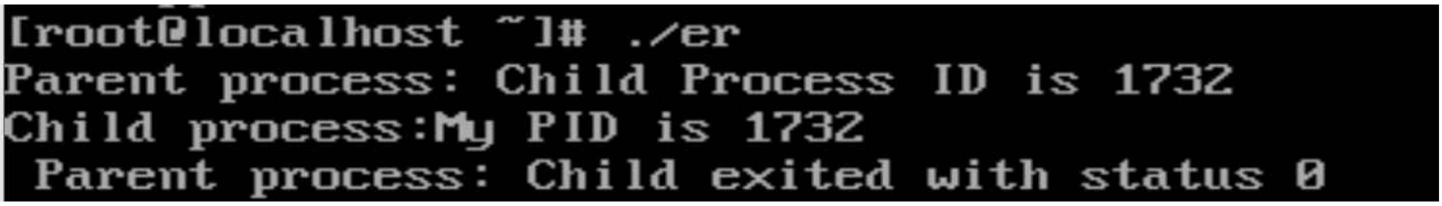
编译代码

```
g++ er.cpp -o er
```

运行代码

```
./er
```

得到结果如下:



```
[root@localhost ~]# ./er
Parent process: Child Process ID is 1732
Child process:My PID is 1732
 Parent process: Child exited with status 0
```

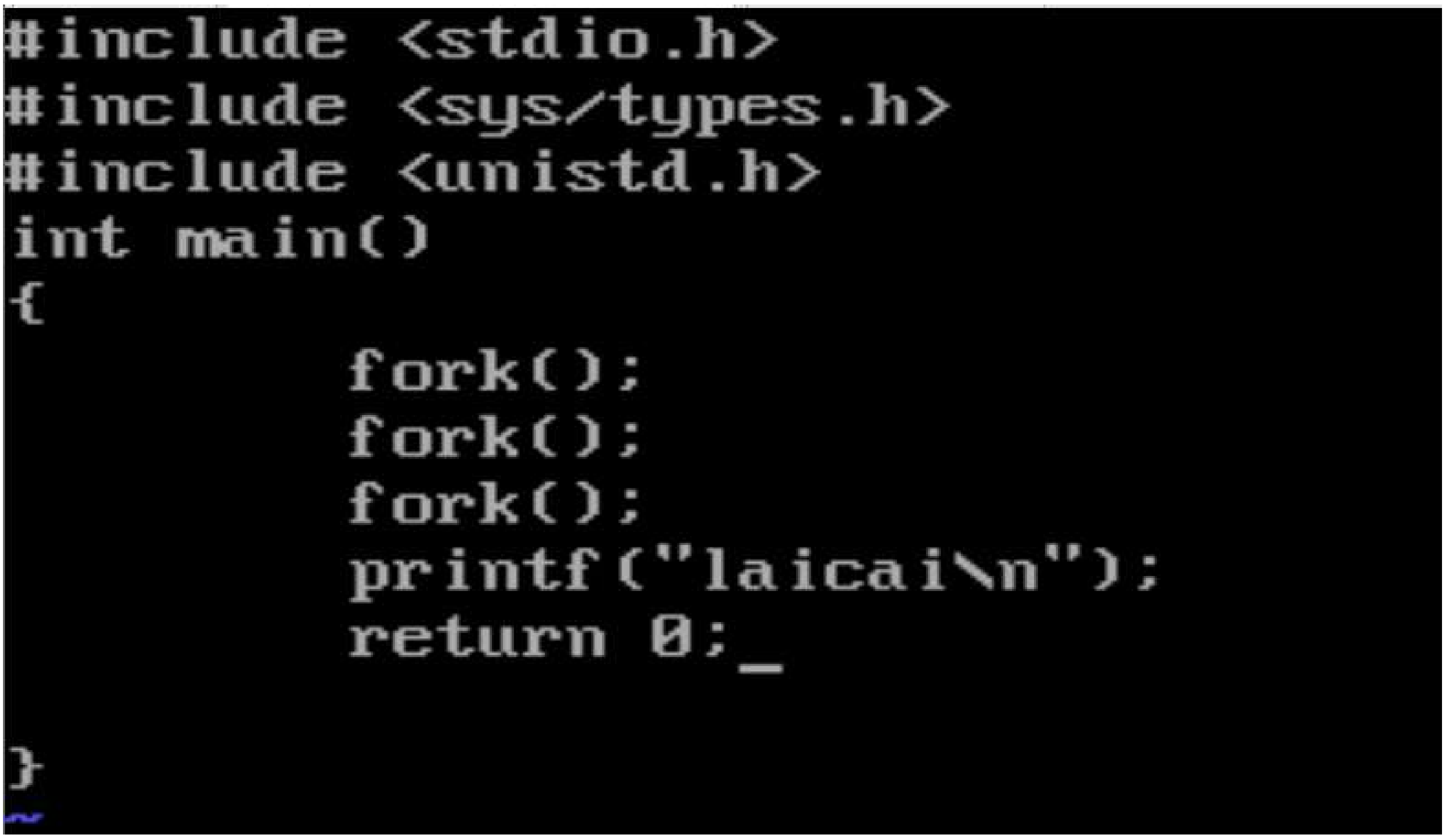
父进程在调用 `waitpid()` 后进入等待状态，知道子进程正常退出以后继续执行代码

多次fork()进程创建实验

1.编写代码

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
    fork();
    fork();
    fork();
    printf("laicai\n");
    return 0;
}
```



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        fork();
        fork();
        printf("laicai\n");
        return 0;_

}
~
```

2.创建结果保存文件

创建结果保存文件demo318

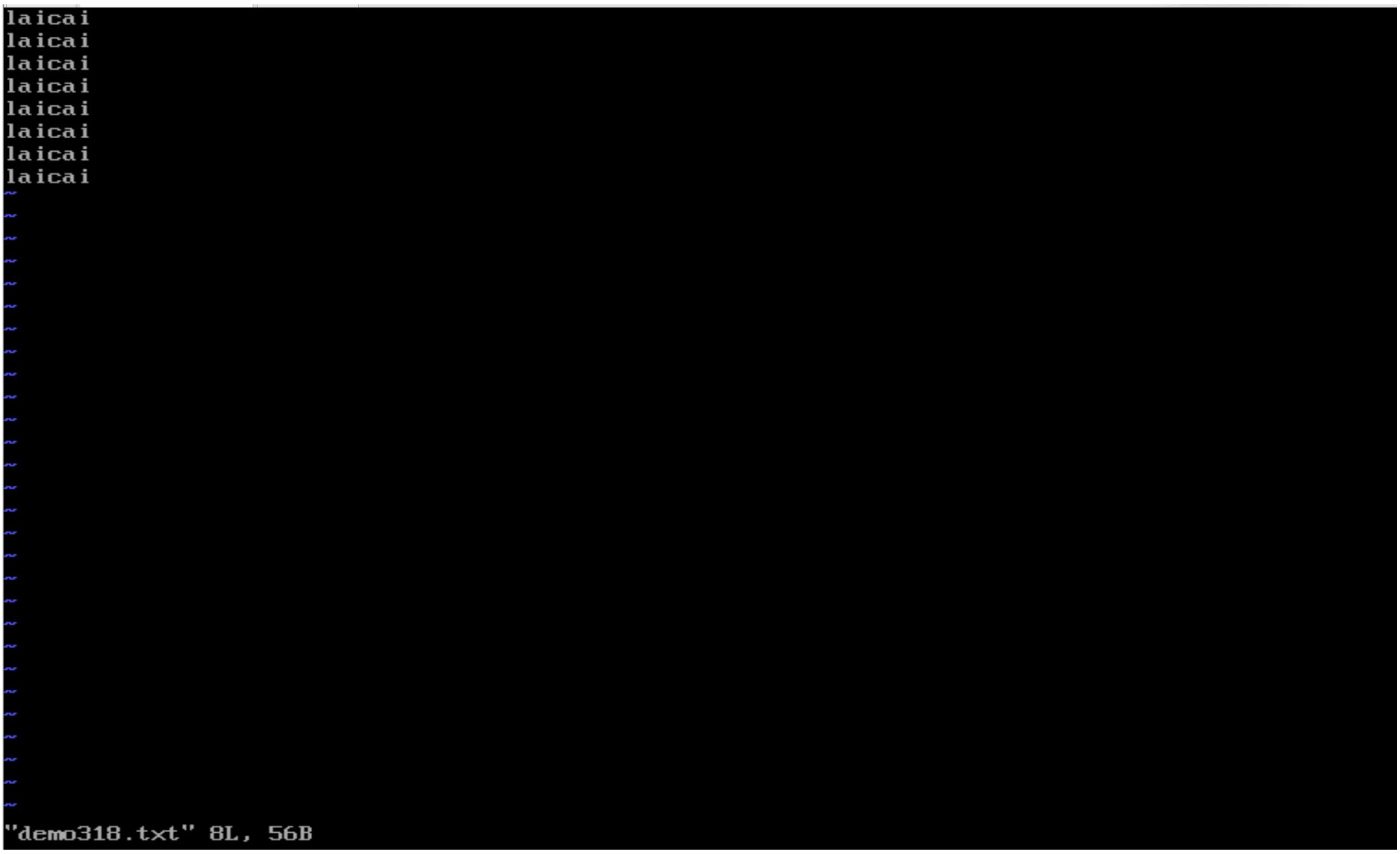
```
touch demo318.txt
```

3.编译并将结果导入到txt文件

```
g++ laicai.cpp -o laicai

./laicai > demo318.txt
```

得到结果如下:



```
laicai
laicai
laicai
laicai
laicai
laicai
laicai
laicai
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"demo318.txt" 8L, 56B
```

多次调用 fork() 函数会以指数形式创建进程
第一次 fork() 以后两个进程
第二次 fork() 以后四个进程
第三次 fork() 以后八个进程
……
每次使用 fork() 以后都会将每个进程复制一遍

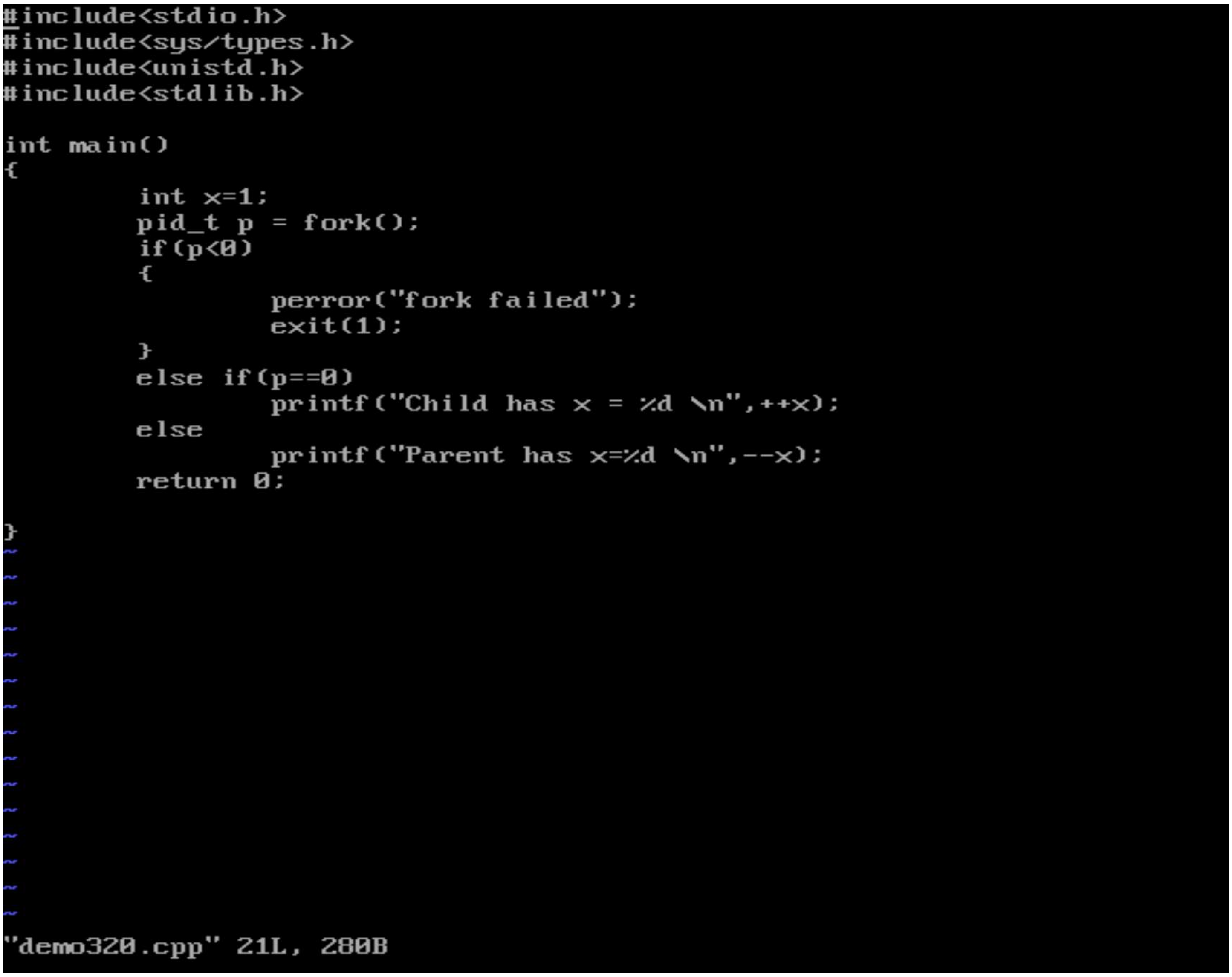
进程独立性实验

1.编写代码

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    int x = 1;
    pid_t p = fork();
    if (p < 0)
    {
        perror("fork fail");
        exit(1);
    }
    else if (p == 0)
        printf("Child has x = %d \n", ++x);
    else
        printf("Parent has x = %d\n", --x);

    return 0;
}
```

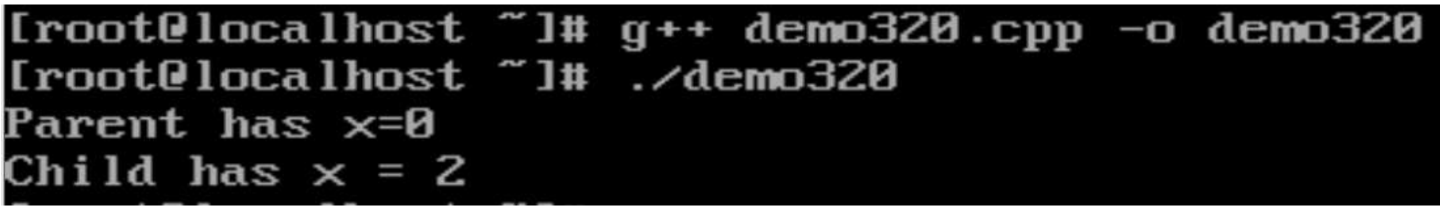


```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
        int x=1;
        pid_t p = fork();
        if(p<0)
        {
                perror("fork failed");
                exit(1);
        }
        else if(p==0)
                printf("Child has x = %d \n",++x);
        else
                printf("Parent has x=%d \n",--x);
        return 0;

}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"demo320.cpp" 21L, 280B
```

2.运行代码



```
[root@localhost ~]# g++ demo320.cpp -o demo320
[root@localhost ~]# ./demo320
Parent has x=0
Child has x = 2
```

这表明父子进程拥有独立的内存空间